南京大学
NANJING UNIVERSITY

存储器层次结构

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

存储器层次结构

- 存储器概述
- 半导体随机存取存储器
- 外部辅助存储器
- 存储器的数据校验
- **高速缓冲存储器**
- 虚拟存储器

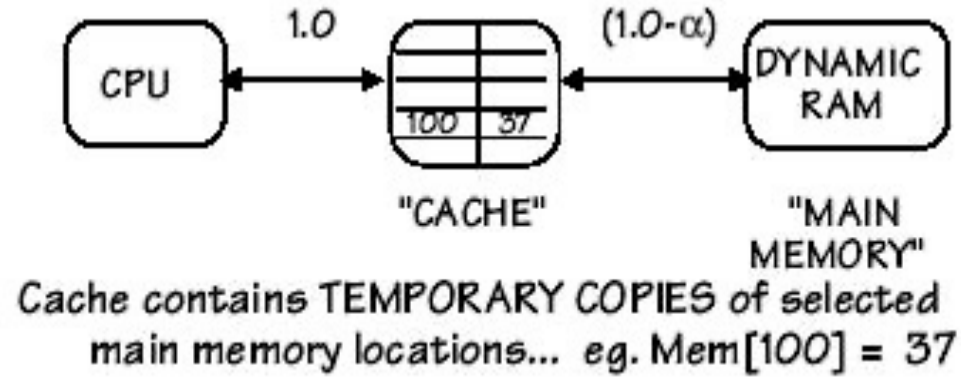
命中率、缺失率、缺失损失

- **命中 Hit: 要访问的信息在Cache中**
  - **Hit Rate(命中率)**：在Cache中的概率
  - **Hit Time (命中时间)** ：在Cache中的访问时间，包括：

    Time to determine hit/miss + Cache access time

    (即： 判断时间 + Cache访问)
- **缺失 Miss: 要找的信息不在Cache中**
  - **Miss Rate (缺失率)** = 1 - (Hit Rate)
  - **Miss Penalty (缺失损失)**：访问一个主存块所花时间
- 命中时间 Hit Time << 缺失损失 Miss Penalty （为什么?）

平均访问时间



- 命中率：α
- 缺失率：1-α
- **平均访问时间：$t_{avg}= \alpha t_c+(1- \alpha)(t_c+t_m)=t_c+(1- \alpha)t_m$**

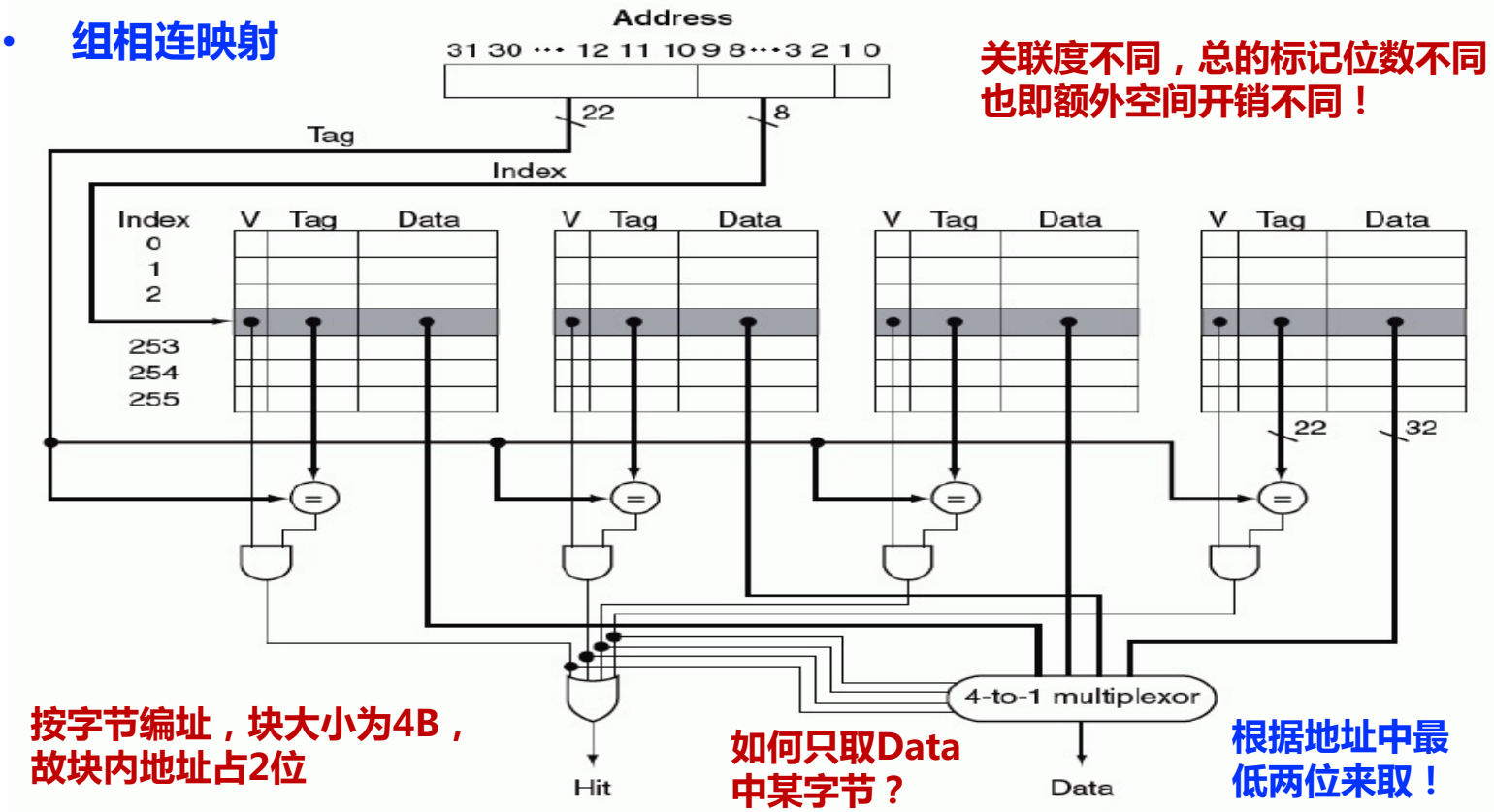
Cache访问时间　　主存访问时间

三种映射方式的比较

- **三种映射方式**
  - **直接映射**：唯一映射（只有一个可能的位置）
  - **全相联映射**：任意映射（每个位置都可能）
  - **N-路组相联映射**：N-路映射（有N个可能的位置）
- **关联度**
  - 一个主存块映射到Cache中时，可能存放的位置个数
    - **直接映射**关联度：**1**，关联度最低。
    - **全相联映射**关联度：**Cache行数**，关联度最高。
    - **N-路组相联映射**关联度：**N**，关联度居中。
- **关联度、缺失率、命中时间**
  - 直观上，你的结论是什么？（ Cache大小和块大小一定时 ）
    - **缺失率**：直接映射最高，全相联映射最低
    - **命中时间**：直接映射最小，全相联映射最大

标记位大小与关联度

- 组相连映射



关联度不同，总的标记位数不同，
也即额外空间开销不同！

按字节编址，块大小为4B，
故块内地址占2位

如何只取Data
中某字节？

根据地址中最
低两位来取！

标记位大小与关联度

- 问题：若主存地址32位，块大小为16字节，Cache总大小为4K行，标记位的总位数是多少？
- 直接映射方式下：相当于每组1行，共4K组，标志占32-4-12=16位，总位数占4Kx16=64K位
- 关联度增加到2倍(2-way)：每组2行，共2K组，标志占32-4-11=17位，总位数占4Kx17=68K位
- 关联度增加到4倍(4-way)：每组4行，共1K组，标志占32-4-10=18位，总位数占4Kx18=72K位
- 全相联时：整个为1组，每组4K行，标志占32-4=28位，总位数占4Kx28=112K位

关联度越高，总的标记位数越多，额外空间开销越大！

Cache中主存块的替换算法

- **问题举例：**

  组相联映射时，假定第0组的两行分别被主存第0和8块占满，此时若需调入主存第16块，根据映射关系，它只能放到Cache第0组，因此，第0组中必须调出一块，那么调出哪一块呢？这就是淘汰策略问题，也称替换算法。

- **常用替换算法有**：
  - 先进先出FIFO（first-in-first-out）
  - 最近最少用LRU（ least-recently used）
  - 最不经常用LFU（ least-frequently used）
  - 随机替换算法（Random）

  等等

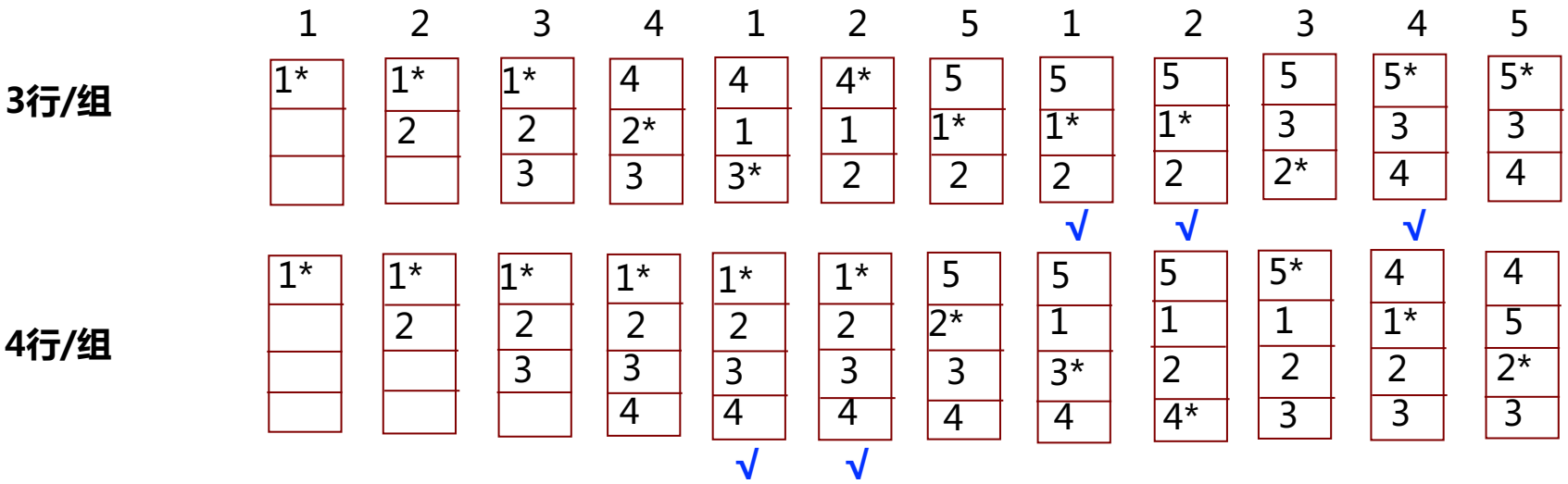
这里的替换策略和后面的虚拟存储器所用的替换策略类似，将是以后操作系统课程的重要内容，本课程只做简单介绍。有兴趣的同学可以进一步自己学习。

先进先出算法FIFO

- **总是把最先进入的那一块淘汰掉。**

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组的情况。

注：通常一组中含有2^k行，这里3行/组主要为了简化问题而假设



由此可见，FIFO不是一种栈算法，即命中率并不随组的增大而提高。

南京大学 NANJING UNIVERSITY

最近最少用算法LRU

- **总是把最近最少用的那一块淘汰掉。**

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、4行/组、 5行/组的情况。

1	2	3	4	1	2	5	1	2	3	4	5
1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4
		1	2	3	4	1	2	5	1	2	3
			1	2	3	4	3	4	5	1	2
						3		3	4	5	1

3行/组							√	√				
4行/组				√	√		√	√				
5行/组				√	√		√	√	√	√	√	

最近最少用算法、最不经常用算法、随机替换算法

- **最近最少用算法LRU**是一种栈算法，它的**命中率随组的增大而提高**。
- 当分块局部化范围(即：某段时间集中访问的存储区)超过了Cache存储容量时，命中率变得很低。极端情况下，假设地址流是1,2,3,4,1 2,3,4,1,……，而Cache每组只有3行，那么，不管是FIFO，还是LRU算法，其命中率都为0。这种现象称为颠簸(Thrashing / PingPong)。
- LRU具体实现时，并不是通过移动块来实现的，而是通过给每个cache行设定一个计数器，根据计数值来记录这些主存块的使用情况。这个计数值称为**LRU位**。

- **最不经常用算法LFU**：也用与每个行相关的计数器来实现，有LRU算法类似，但不完全相同。
- **随机替换算法**：从候选行的主存块中随机选取一个淘汰掉，与使用情况无关。（在性能上只稍逊于基于使用情况的算法，而且代价低。）

最近最少用算法LRU

- **通过计数值来确定cache行中主存块的使用情况**

即：计数值为0的行中的主存块最常被访问，计数值为3的行中的主存块最不经常被访问，先被淘汰！

- **计数器变化规则：**
  - 每组4行时，计数器有2位。计数值越小则说明越被常用。
  - 命中时，被访问行的计数器置0，比其低的计数器加1，其余不变。
  - 未命中且该组未满时，新行计数器置为0，其余全加1。
  - 未命中且该组已满时，计数值为3的那一行中的主存块被淘汰，新行计数器置为0，其余加1。

1		2		3		4		1		2		5		1		2		3		4		5	
0	1	1	1	2	1	3	1	0	1	1	1	2	1	0	1	1	1	2	1	3	1	0	5
		0	2	1	2	2	2	3	2	0	2	1	2	2	2	0	2	1	2	2	2	3	4
				0	3	1	3	2	3	3	3	0	5	1	5	2	5	3	5	0	4	2	3
						0	4	1	4	2	4	3	4	3	4	3	4	0	3	1	3	1	2

何时需要替换

- **直接映射:**
  - 映射唯一，毫无选择，无需考虑替换
- **全相连映射:**
  - 每个主存数据可存放到Cache任意行中，需考虑替换
- **N-路组相连映射:**
  - 每个主存数据有N个Cache行可选择，需考虑替换
- **对于全相连映射和N-路组相连映射，可能需要替换。其过程为：**
  - 从主存取出一个新块
  - 选择一个有映射关系的空Cache行
  - 若对应Cache行被占满时又需调入新主存块，则必须考虑从Cache行中替换出一个主存块

替换算法举例

- 假定计算机系统主存空间大小为32Kx16位，且有一个4K字的**4路组相联Cache**，主存和Cache之间的数据交换块的大小为64字。假定Cache开始为空，处理器顺序地从存储单元0、1、…、4351中取数，一共重复10次。设Cache比主存快10倍。采用**LRU算法**。试分析Cache的结构和主存地址的划分。说明采用Cache后速度提高了多少？

- 答：假定主存按字编址。每字16位。

  主存：32K字=512块 x 64字 / 块

  Cache：4K字=16组 x 4行 / 组 x 64 字 / 行

  主存地址划分为：

标志位	组号	字号
5	4	6

4352/64=68，所以访问过程实际上是对前68块连续访问10次。

替换算法举例

	第0 行	第1 行	第2 行	第3 行
第0组	0/64/48	16/0/64	32/16	48/32
第1组	1/65/49	17/1/65	33/17	49/33
第2组	2/66/50	18/2/66	34/18	50/34
第3组	3/67/51	19/3/67	35/19	51/35
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

LRU算法：第一次循环，每一块只有第一字未命中，其余都命中；以后9次循环，有20块的第一字未命中，其余都命中。

所以，命中率p为 (43520-68-9x20)/43520=99.43%.

速度提高：tm/ta=tm/(tc+(1-p)tm)=10/(1+10x(1-p))=9.5倍.

Cache的一致性问题

- **为何要保持在Cache和主存中数据的一致？**
  - 因为Cache中的内容是主存块副本，当对Cache中的内容进行更新时，就存在Cache和主存如何保持一致的问题。
  - 以下情况也会出现“Cache一致性问题”
    - **当多个设备都允许访问主存时**

      例如：I/O设备可直接读写内存时，如果Cache中的内容被修改，则I/O设备读出的对应主存单元的内容无效；若I/O设备修改了主存单元的内容，则Cache中对应的内容无效。
    - **当多个CPU都带有各自的Cache而共享主存时**

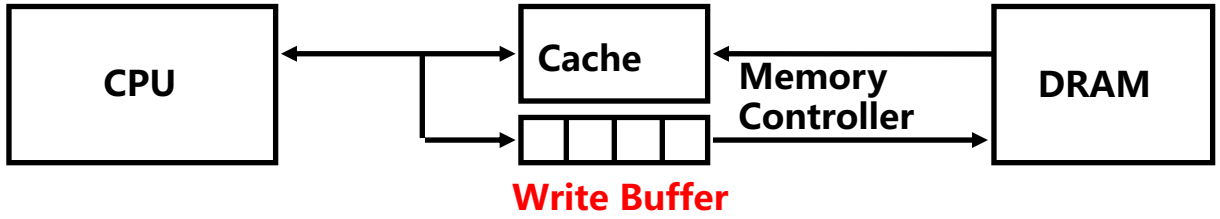
      某个CPU修改了自身Cache中的内容，则对应的主存单元和其他CPU中对应的内容都变为无效。
- **写操作有两种情况**
  - **写命中（Write Hit）**：要写的单元已经在Cache中
  - **写不命中（Write Miss）**：要写的单元不在Cache中

Cache的一致性问题

- **处理Cache读比Cache写更容易，故指令Cache比数据Cache容易设计**
- **对于写命中，有两种处理方式**
  - **全写法 Write Through** (通过式写、写直达、直写)
    - 同时写Cache和主存单元
      - **假定一次写主存需要100个CPU时钟周期，10%的存储指令使CPI增加到：1.0+100x10%=11**
    - 使用写缓冲（Write Buffer）
  - **回写法 Write Back** (一次性写、写回、回写)
    - 只写cache不写主存，缺失时一次写回，每行有个修改位（“dirty bit-脏位”），大大降低主存带宽需求，控制可能很复杂
- **对于写不命中，有两种处理方式**
  - **写分配 Write Allocate**
    - 将主存块装入Cache，然后更新相应单元
    - 试图利用空间局部性，但每次都要从主存读一个块
  - **非写分配 Not Write Allocate**
    - 直接写主存单元，不把主存块装入到Cache

- ✓ **直写Cache可用非写分配或写分配**
- ✓ **回写Cache通常用写分配**

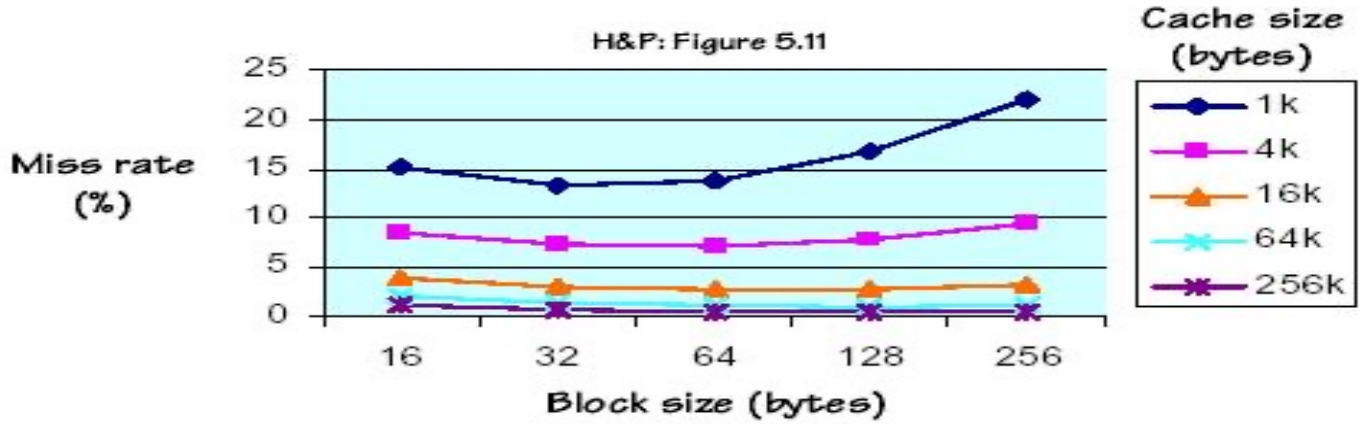
全写法：写缓冲



- **在 Cache 和 Memory之间加一个Write Buffer**
  - CPU同时写数据到Cache和Write Buffer
  - Memory controller（存控）将缓冲内容写主存
- **Write buffer (写缓冲) 是一个FIFO队列**
  - 一般有4项
  - 在存数频率不高时效果好
- **最棘手的问题**
  - 当频繁写时，易使写缓存饱和，发生阻塞
- **如何解决写缓冲饱和？**
  - 加一个二级Cache
  - 使用Write Back方式的Cache

Cache大小、Block大小和缺失率的关系

- Cache性能由缺失率确定，而缺失率与Cache大小、Block大小等有关



- spatial locality: larger blocks → reduce miss rate
- fixed cache size: larger blocks
  → fewer lines in cache
  → higher miss rate, especially in small caches

- Cache大小：Cache越大，Miss率越低，但成本越高！
- Block大小：Block大小与Cache大小有关，且不能太大，也不能太小！

Cache和程序性能

- **程序的性能**指执行程序所用的时间
- **程序执行所用时间**与程序执行时访问指令和数据所用的时间有很大关系，而指令和数据的访问时间与cache命中率、命中时间和缺失损失有关
- 对于给定的计算机系统而言，命中时间和缺失损失是确定的，因此，**指令和数据的访存时间**主要由cache命中率决定
- **Cache命中率**主要由程序的空间局部性和时间局部性决定。因此，为了提高程序的性能，程序员须编写出具有良好访问局部性的程序
- 考虑**程序的访问局部性**通常在数据的访问局部性上下工夫
- **数据的访问局部性**主要是指数组、结构等类型数据访问时的局部性，这些数据结构的数据元素访问通常是通过循环语句进行的，所以，如何合理地处理循环对于数据访问局部性来说是非常重要的。

Cache和程序性能举例

- 举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i, j, sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。

程序 A：

```
int a[256][256];
……
int sum_array1 ( )
{
        int i, j, sum = 0;
        for ( i = 0; i < 256; i++)
                for (j = 0; j < 256; j++)
                        sum += a[i] [j];
        return sum;
}
```

程序 B：

```
int a[256][256];
……
int sum_array2 ( )
{
        int i, j, sum = 0;
        for ( j = 0; j < 256; j++)
                for ( i = 0; i < 256; i++)
                        sum += a[i] [j];
        return sum;
 }
```

（1）不考虑用于一致性和替换的控制位，**数据cache的总容量为多少**？

（2）a[0][31]和a[1][1]各自所在主存块对应的**cache行号分别是多少**？

（3）程序A和B的**数据访问命中率各是多少**？哪个程序的执行时间更短？

Cache和程序性能举例

- **举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i, j, sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。**

（1）主存地址空间大小为256MB，因而主存地址为28位，其中6位为块内地址，3位为cache行号（行索引），标志信息有28-6-3=19位。在不考虑用于cache一致性维护和替换算法的控制位的情况下，**数据cache的总容量为**：8×(19+1+64×8)=4256位=532字节 。

（2）**a[0][31]**的地址为320+4×31=444，[444/64]=6（取整），因此a[0][31]对应的主存块号为6。6 mod 8=6，**对应cache行号为6**。

或: 444=0000 0000 0000 0000 000 110 111100B，中间3位110为行号（行索引），因此，对应的cache行号为6。

a[1][1]对应的cache行号为：[(320+4×(1×256+1))/64] mod 8=**5**。

（3）**A中**数组访问顺序与存放顺序相同，共访问64K次，占4K个主存块；首地址位于一个主存块开始，故每个主存块总是第一个元素缺失，其他都命中，共缺失4K次，**命中率为1-4K/64K=93.75%**。

方法二：每个主存块的命中情况一样。对于一个主存块，包含16个元素，需访存16次，其中第一次不命中，因而**命中率为15/16=93.75%**。

B中访问顺序与存放顺序不同，依次访问的元素分布在相隔256×4=1024的单元处，它们都不在同一个主存块中,cache共8行，一次内循环访问16块，故再次访问同一块时，已被调出cache，因而每次都缺失，**命中率为0**。

Cache设计应考虑的问题

- 刚引入Cache时只有一个Cache。近年来多Cache系统成为主流
- **多Cache系统中，需考虑两个方面**：
  - **[1] 单级/多级？**

  外部(Off-chip)Cache： 不做在CPU内而是独立设置一个Cache

  片内(On-chip)Cache： 将Cache和CPU作在一个芯片上

  单级Cache：只用一个片内Cache

  多级Cache：同时使用L1 Cache和L2 Cache，甚至有L3 Cache，L1 Cache更靠近CPU，其速度比L2快，其容量比L2小
  - **[2] 联合/分立？**

  分立：指数据和指令分开存放在各自的数据和指令Cache中

  一般L1 Cache都是分立Cache，为什么？

  L1 Cache的命中时间比命中率更重要！为什么？

  联合：指数据和指令都放在一个Cache中

  一般L2 Cache都是联合Cache，为什么？

  L2 Cache的命中率比命中时间更重要！为什么？

因为缺失时需从主存取数，并要送L1和L2cache，损失大！

多级Cache的性能

- 采用L2 Cache的系统，其缺失损失的计算如下：
  - 若L2 Cache包含所请求信息，则缺失损失为L2 Cache访问时间
  - 否则访问主存，并取到L1 Cache和L2 Cache（缺失损失更大）
- 例：某处理器在无cache缺失时CPI为1，时钟频率为5GHz。假定访问一次主存的时间（包括所有的缺失处理）为100ns，平均每条指令在L1 Cache中的缺失率为2%。若增加一个L2 Cache，其访问时间为5ns，而且容量足够大到使全局缺失率减为0.5%，问处理器执行指令的速度提高了多少？
  - 解：**如果只有一级Cache**，则缺失只有一种。即L1缺失(需访问主存)，其缺失损失为：100nsx5GHz=500个时钟，CPI=1+500x2%=11.0。

    如果有二级Cache，则有两种缺失：

    L1缺失(需访问L2 Cache)：5nsx5GHz=25个时钟

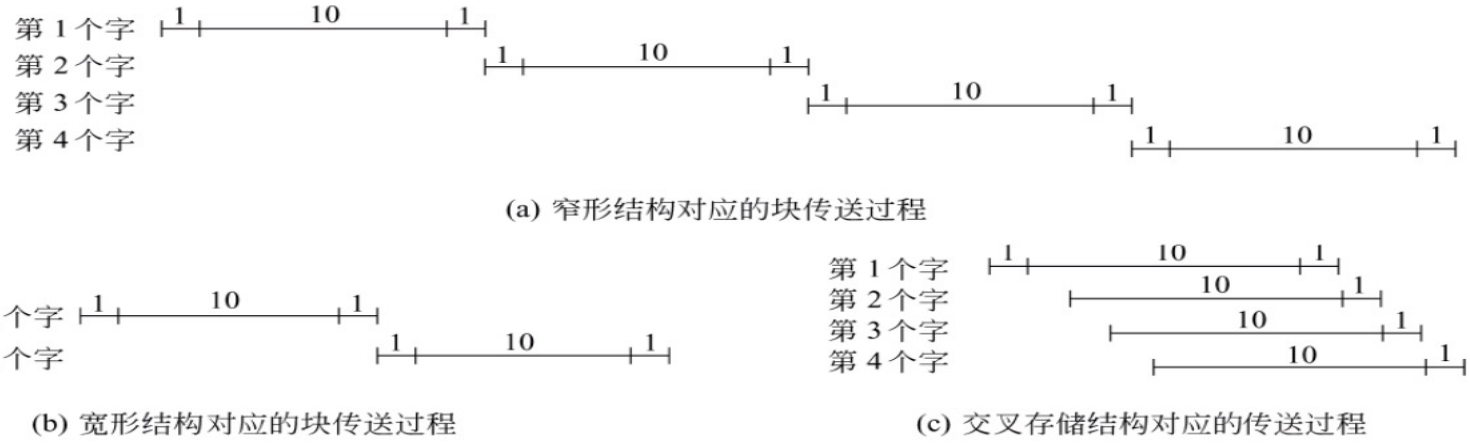
    L1和L2都缺失(需访问主存)：500个时钟

    因此，CPI=1+25x2%+500x0.5%=4.0

    二者的性能比为11.0/4.0=2.8倍！

主存-总线-Cache间的连接结构问题

- **CPU从主存取一块信息到Cache**：发送地址和读命令到主存(1个周期)、主存准备好一个数据(10个周期)、从总线传送一个数据(1个周期)。



(a) 窄形结构对应的块传送过程

(b) 宽形结构对应的块传送过程

(c) 交叉存储结构对应的传送过程

图 7.35　主存块在主存-总线-cache 之间的传送过程

- **主存、总线、cache之间有三种连接方式：**
  - **窄形结构**：每次传送一个字
  - **宽形结构**：每次传送多个字
  - **多模块交叉存取结构**：轮流启动多个存储模块进行读写，按一个字的宽度进行传送。

DRAM结构、总线事务类型与Cache的配合问题

- 如何合理设计DRAM就结构，如何使存储器总线在一次总线事务中高效地传输一个主存块？

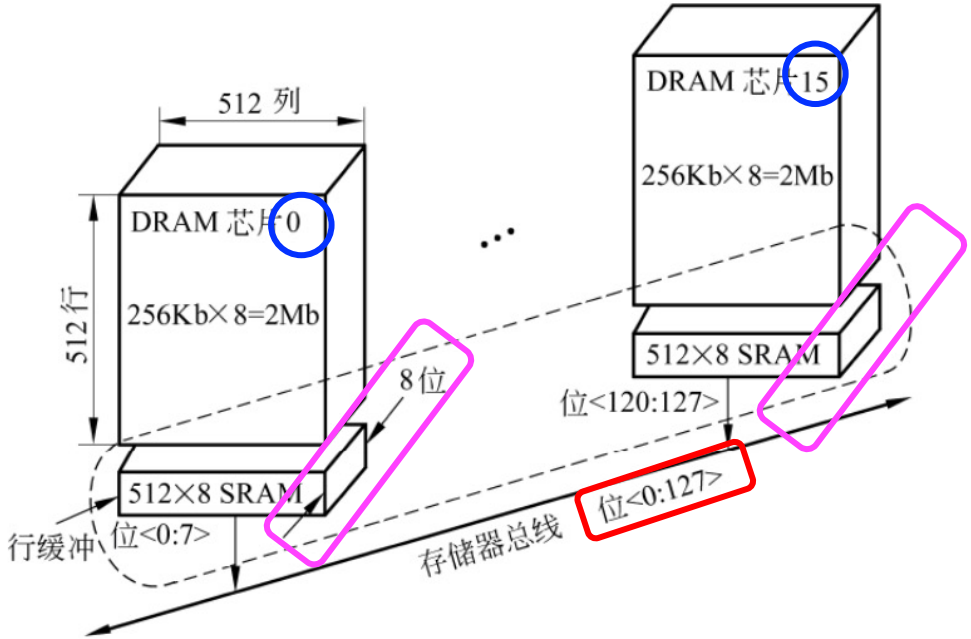


图 7.36　内存条中芯片排列示意

存储器层次结构

- 存储器概述
- 半导体随机存取存储器
- 外部辅助存储器
- 存储器的数据校验
- 高速缓冲存储器
- **虚拟存储器**

虚拟存储器

- **虚拟存储技术的引入用来解决一对矛盾**
  - 一方面，由于技术和成本等原因，主存容量受到限制
  - 另一方面，系统程序和应用程序要求主存容量越来越大
- **虚拟存储技术的实质**
  - 程序员在比实际主存空间大得多的逻辑地址空间中编写程序
  - 程序执行时，把当前需要的程序段和相应的数据块调入主存，其他暂不用的部分存放在磁盘上
  - 指令执行时，通过硬件将逻辑地址（也称虚拟地址或虚地址）转化为物理地址（也称主存地址或实地址）
  - 在发生程序或数据访问失效(缺页)时，由操作系统进行主存和磁盘之间的信息交换
- **虚拟存储器机制由硬件与操作系统共同协作实现**，涉及到操作系统中的许多概念，如进程、进程的上下文切换、存储器分配、虚拟地址空间、缺页处理等。

虚拟存储机制

- CPU通过**存储器管理部件(MMU)**将指令中的**逻辑地址**(也称虚拟地址或虚地址，VA)转换为主存的**物理地址**(也称为主存地址或实地址，PA)。

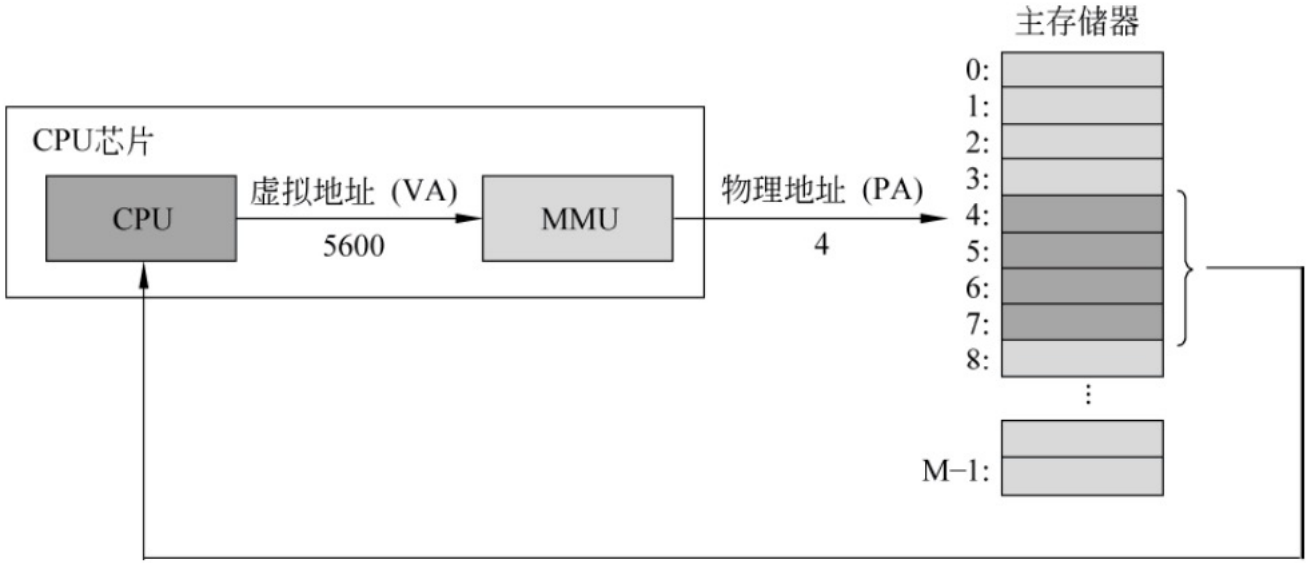
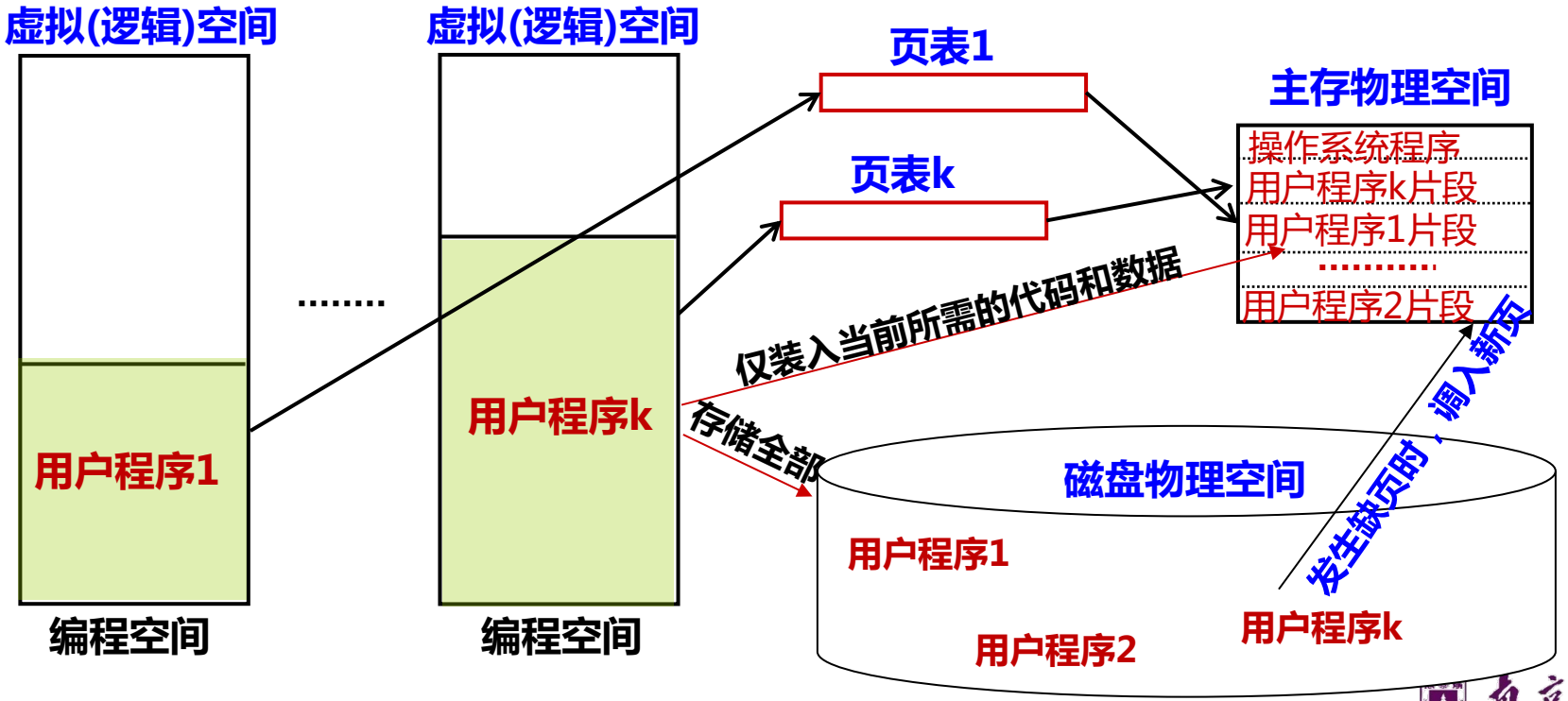


图 7.38　具有虚拟存储机制的 CPU 和主存的连接

虚拟存储技术的实质

通过页表建立虚拟空间和物理空间之间的映射!



虚拟地址空间

- Linux在X86上的虚拟地址空间（其他Unix系统的设计类此）
  - **内核空间（Kernel）**
    - 与进程相关的数据结构
    - 物理存储区
    - 内核代码和数据
  - **用户空间**
    - 用户栈（User Stack）
    - 共享库（Shared Libraries）
    - 堆（heap）
    - 可读写数据（Read/Write Data）
    - 只读数据（Read-only Data）
    - 代码（Code）

问题：加载时是否真正从磁盘调入信息到主存？

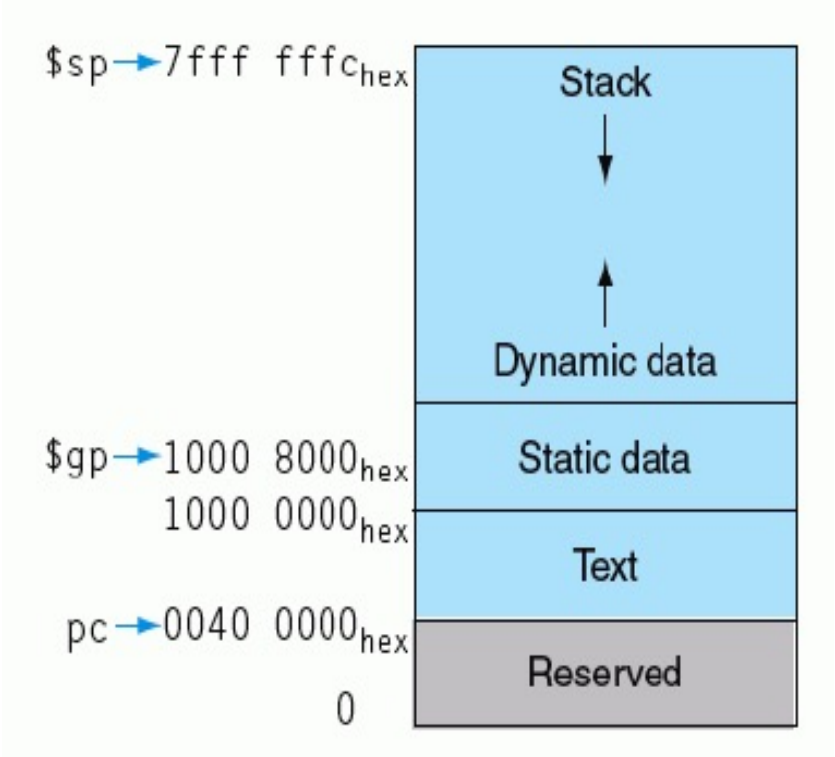
实际上不会从磁盘调入，只是将虚拟页和磁盘上的数据/代码建立对应关系，称为**“映射”**。

图 7.39 进程虚拟地址空间结构

可执行文件相关内容“复制”到代码段和数据段

MIPS程序和数据的存储器分配

- **问题**：你知道一个程序在“编辑、编译、汇编、链接、装入”过程中的哪个环节确定了每条指令及其操作数的虚拟地址吗？
- **链接时确定虚拟地址；装入时生成页表以建立虚拟地址与物理地址之间的映射！**
- 请参考《深入理解计算机系统》和有关Linux内核分析方面的资料。



每个用户程序都有相同的虚拟地址空间！

这就是每个进程的虚拟（逻辑）地址空间！

虚拟存储器管理

- **实现虚拟存储器管理，需考虑：**
  - 块大小（在虚拟存储器中“块”被称为“页 / Page”）应多大？
  - 主存与辅存的空间如何分区管理?
  - 程序块 / 存储块之间如何映像?
  - 逻辑地址和物理地址如何转换，转换速度如何提高?
  - 主存与辅存之间如何进行替换（与Cache所用策略相似）?
  - 页表如何实现，页表项中要记录哪些信息?
  - 如何加快访问页表的速度?
  - 如果要找的内容不在主存，怎么办?
  - 如何保护进程各自的存储区不被其他进程访问?

有三种虚拟存储器实现方式：分页式、分段式、段页式

这些问题是由硬件和OS共同协调解决的！

分页式虚拟存储器

- **与“Cache--主存”层次相比：**
  - **页大小（2KB~64KB）比Cache中的Block大得多！ 为什么？**
  - **采用全相联映射！为什么?**
    - ✓ 因为缺页的开销比Cache缺失开销大的多！缺页时需要访问磁盘（约几百万个时钟周期），而cache缺失时，访问主存仅需几十到几百个时钟周期！因此，页命中率比cache命中率更重要！“大页面”和“全相联”可提高页命中率。
  - **通过软件来处理“缺页”！为什么?**
    - ✓ 缺页时需要访问磁盘（约几百万个时钟周期），慢！不能用硬件实现。
  - **采用Write Back写策略！ 为什么?**
    - ✓ 避免频繁的慢速磁盘访问操作。
  - **地址转换用硬件实现！为什么?**
    - ✓ 加快指令执行

页表结构

- **页表**：建立各个虚拟页（**虚拟地址中**）和所存放的主存页框号（**主存空间中**）或磁盘上存储位置之间的关系
- **页表首址**：记录在页表基址寄存器中

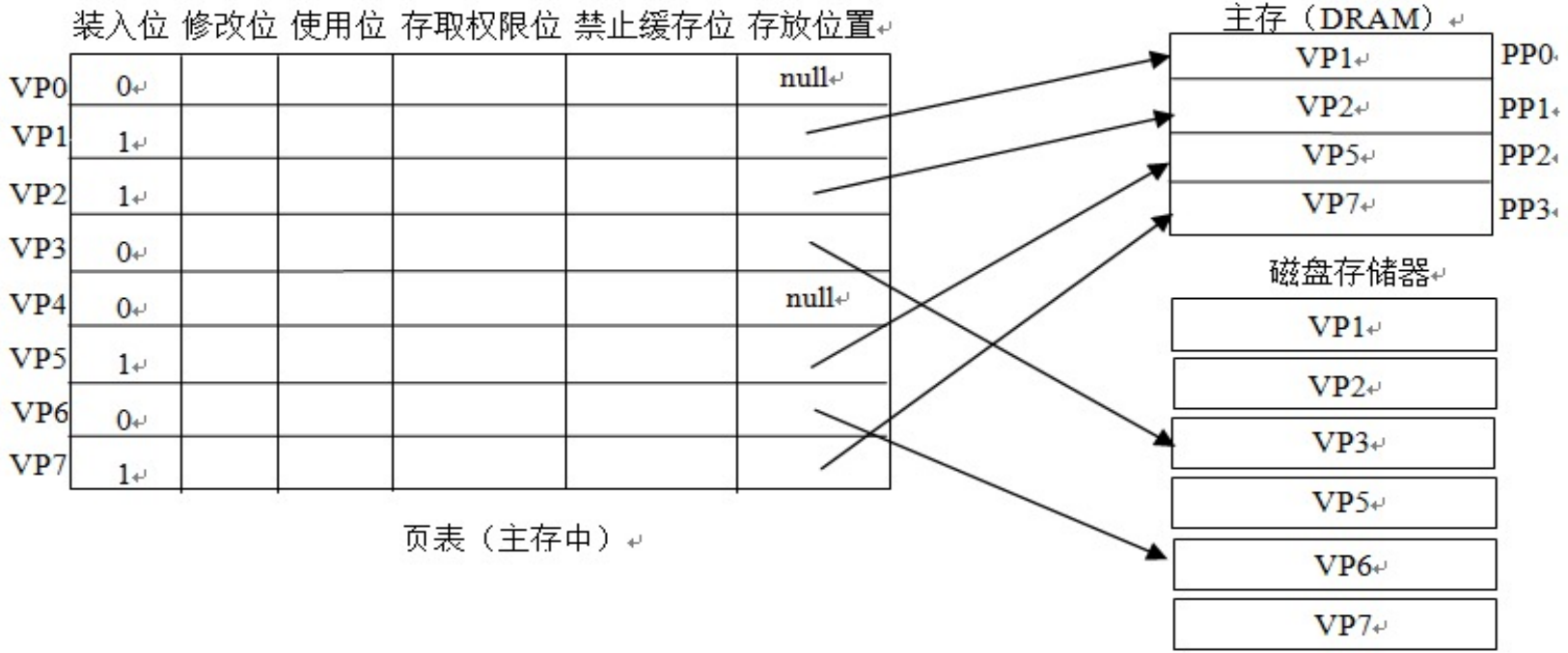
	装入位	修改位	使用位	访问权限位	禁止缓存位	存放位置
VP0	0					null
VP1	1					
VP2	1					
VP3	0					
VP4	0					null
VP5	1					
VP6	0					
VP7	1					

页表（主存中）

- 每个进程有一个页表，其中有**装入位、修改（Dirt）位、替换控制位、访问权限位、禁止缓存位、实页号**。
- 一个页表的项数由什么决定？ **理论上由虚拟地址空间大小决定。**
- 每个进程的页表大小一样吗？

  各进程有相同虚拟空间，故理论上一样。实际大小看具体实现方式，如“null”页面如何处理等

主存中的页表示例



- **未分配页**：进程的虚拟地址空间中“null”对应的页（如VP0、VP4）
- **已分配的缓存页**：有内容对应的已装入主存的页（如VP1、VP2、VP5等）
- **已分配的未缓存页**：有内容对应但未装入主存的页（如VP3、VP6）

逻辑地址转换为物理地址的过程

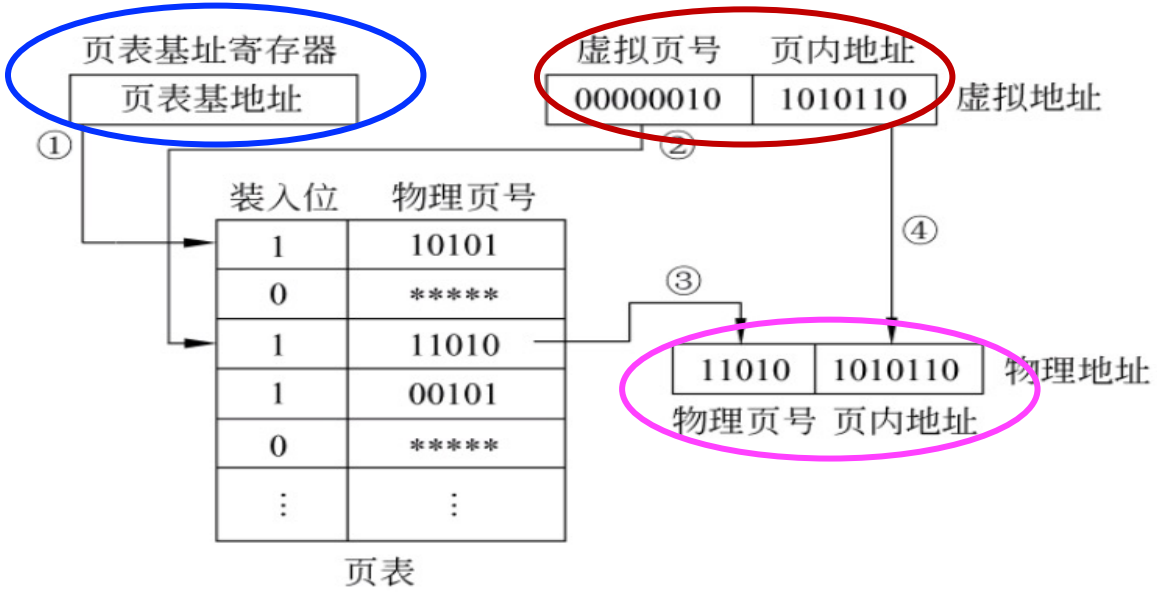


图 7.42　分页式虚存的地址转换

问题：虚拟页与主存页框之间采用全相联方式进行映射，为何不像全相联Cache那样（高位地址是Tag），而高位地址是索引呢？

信息访问中可能出现的异常情况

可能有两种异常情况：

1）缺页（ page fault ）

产生条件：当Valid（有效位 / 装入位）为 0 时

相应处理：从磁盘读到内存，若内存没有空间，则还要从内存选择一页替换到磁盘上，替换算法类似于Cache，采用回写法，淘汰时，根据“dirty”位确定是否要写磁盘

- 当前指令执行被阻塞，当前进程被挂起，处理结束回到原指令继续执行

2）保护违例（ protection_violation_fault ）或访问违例

产生条件： 当Access Rights (存取权限)与所指定的具体操作不相符时

相应处理：在屏幕上显示“内存保护错”或“访问违例”信息

- 当前指令的执行被阻塞，当前进程被终止

Access Rights (存取权限)可能的取值有哪些？

R = Read-only, R/W = read/write, X = execute only

快表

- 把经常要查的页表项放到Cache中，这种在Cache中的页表项组成的页表称为后备转换缓冲器(Translation Lookaside Buffer, TLB)，通常称为快表

- TLB中的页表项：tag+主存页表项

Virtual Address (tag)	Physical Address	Dirty	Ref	Valid	Access
	对应物理页框号				

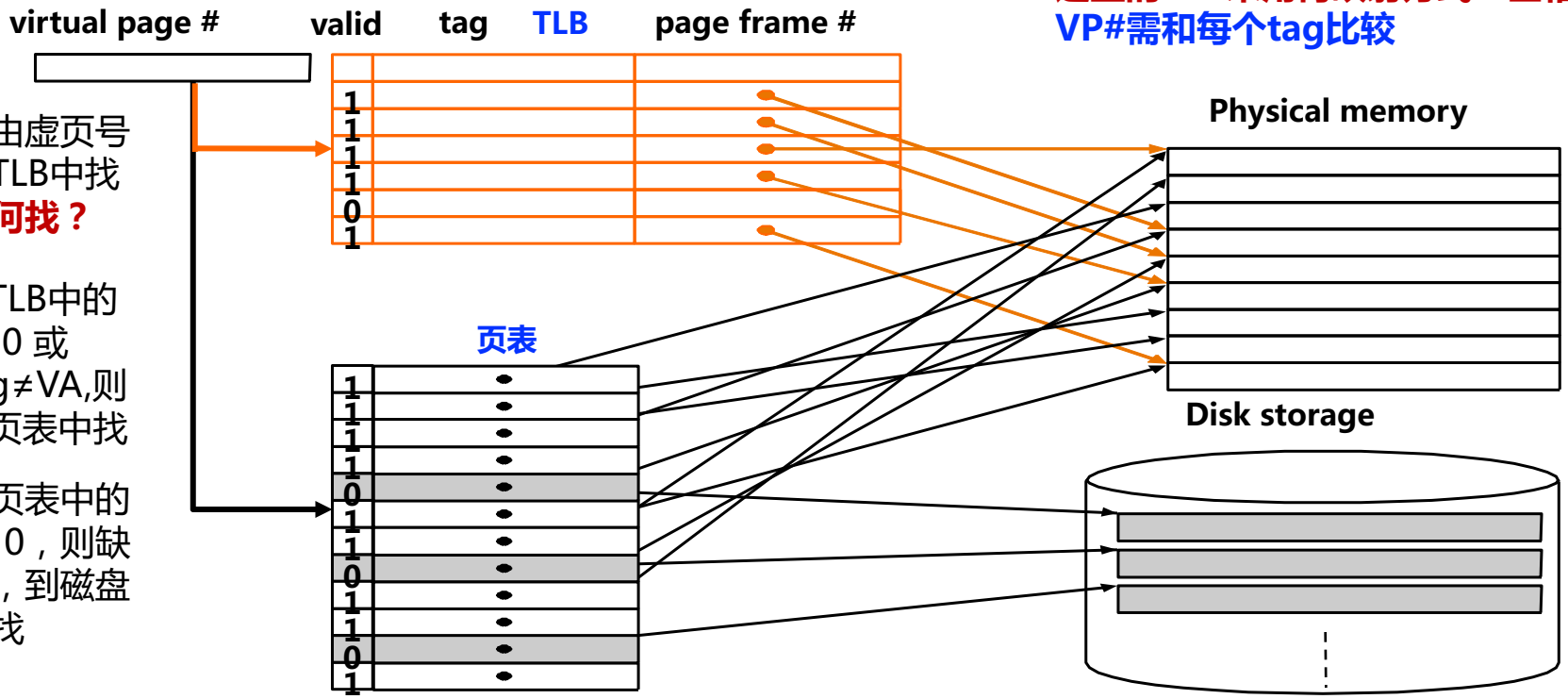
- CPU访存时，地址中虚页号被分成tag+index，tag用于和TLB页表项中的tag比较，index用于定位需要比较的表项

- TLB全相联时，没有index，只有Tag，虚页号需与每个Tag比较；TLB组相联时，则虚页号高位为Tag，低位为index，用作组索引。

快表

这里的TLB采用何映射方式？全相连！
VP#需和每个tag比较

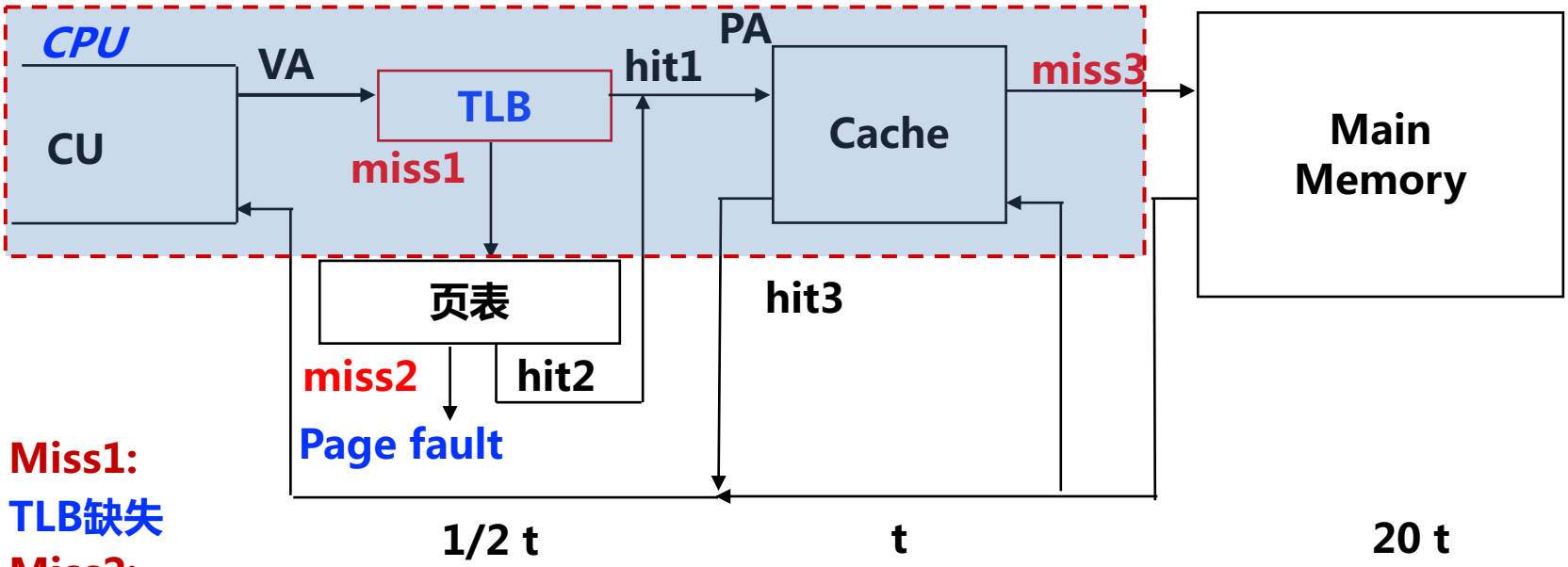
- 先由虚页号到TLB中找
- ➢ **如何找？**
- 若TLB中的V=0 或Tag≠VA,则到页表中找
- 若页表中的V=0，则缺页，到磁盘中找



问题：引入TLB的目的是什么？减少到内存查页表的次数！

主存页表中需要Tag吗？为什么？

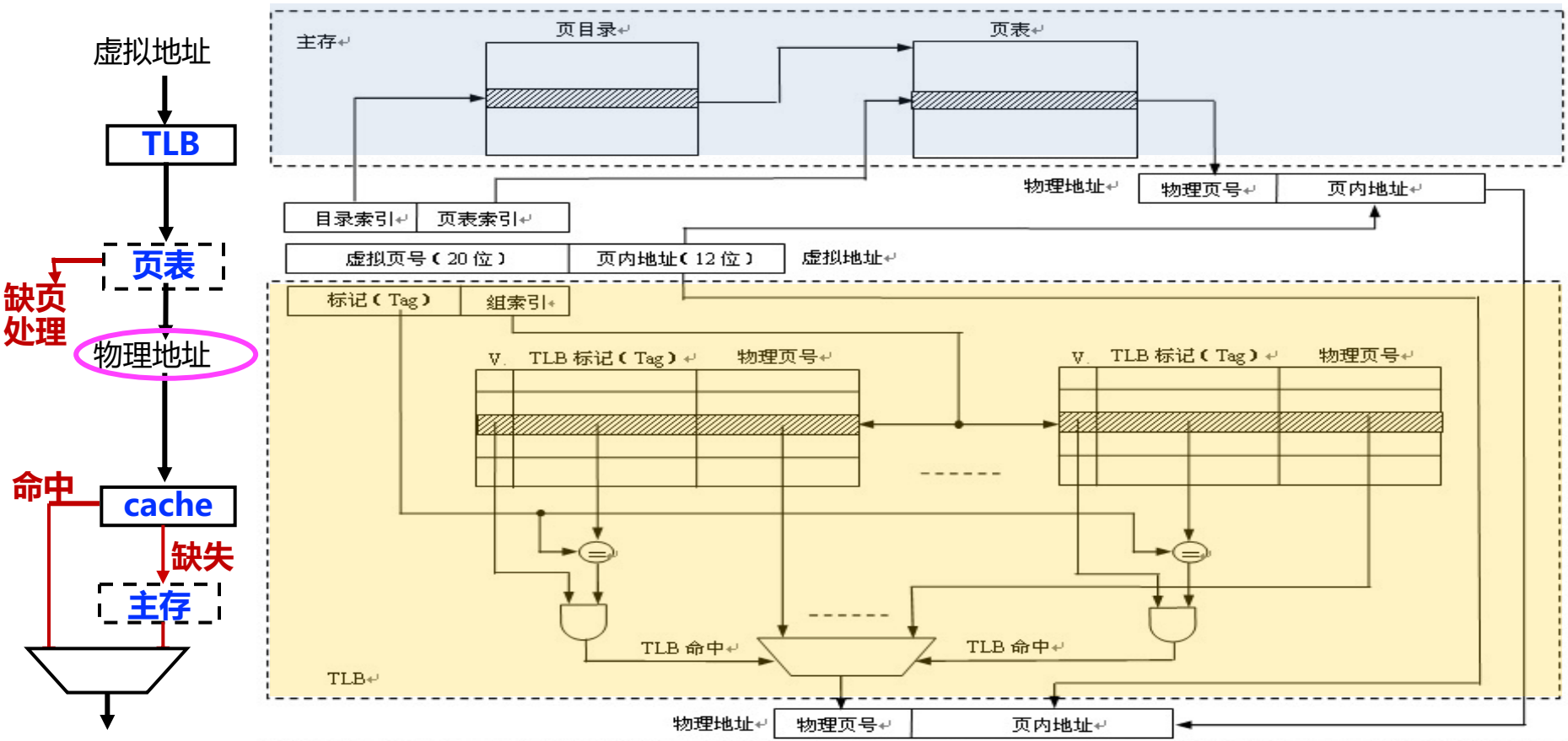
快表



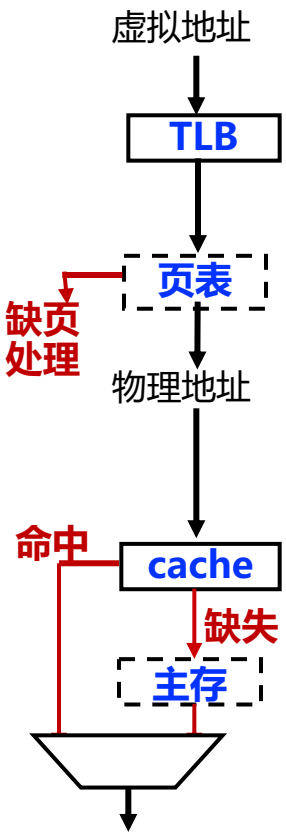
- Miss1:
  - TLB缺失
- Miss2:
  - 缺页
- Miss3:
  - PA 在主存中，但不在Cache中

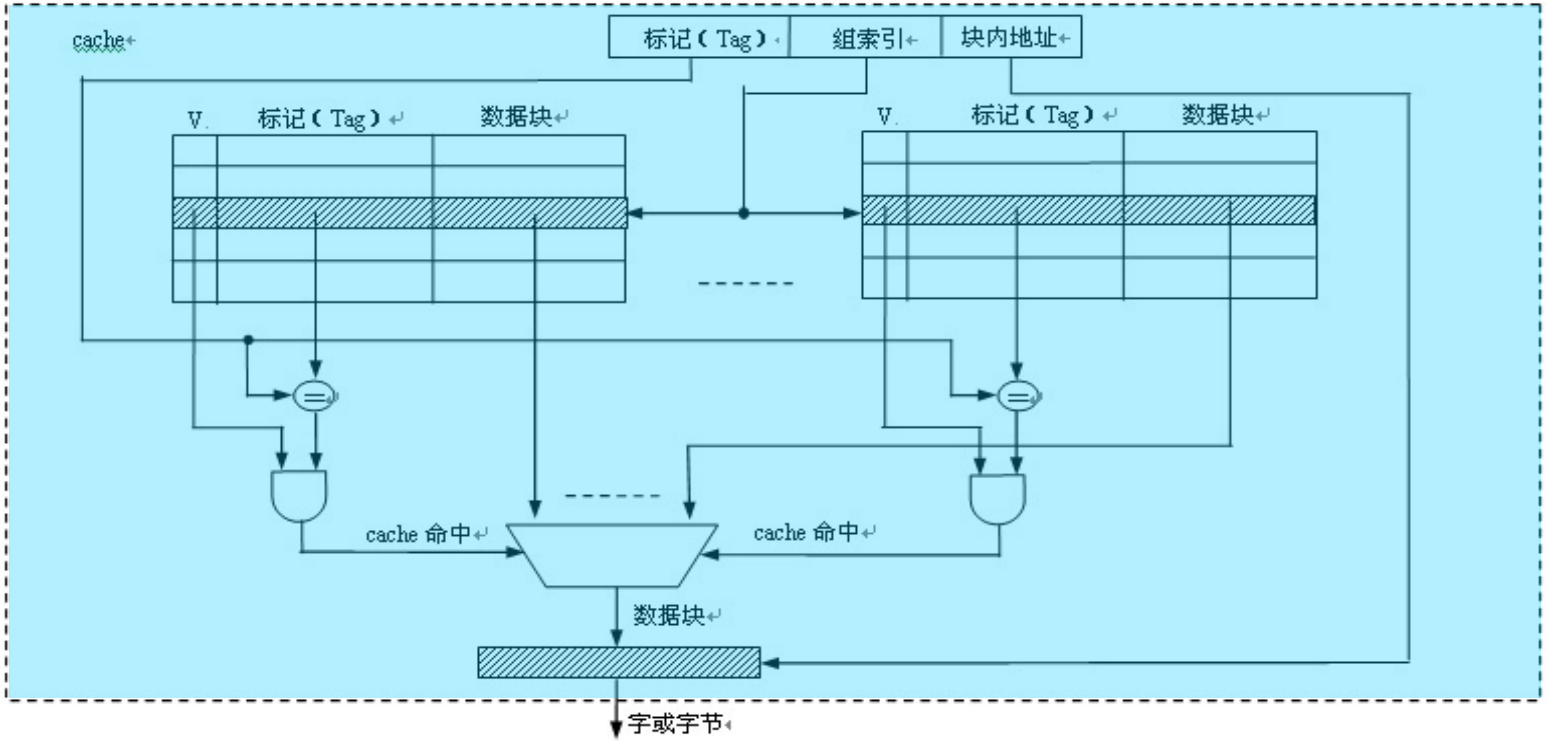
TLB冲刷指令和Cache冲刷指令
都是操作系统使用的特权指令

TLB和Cache的访问过程



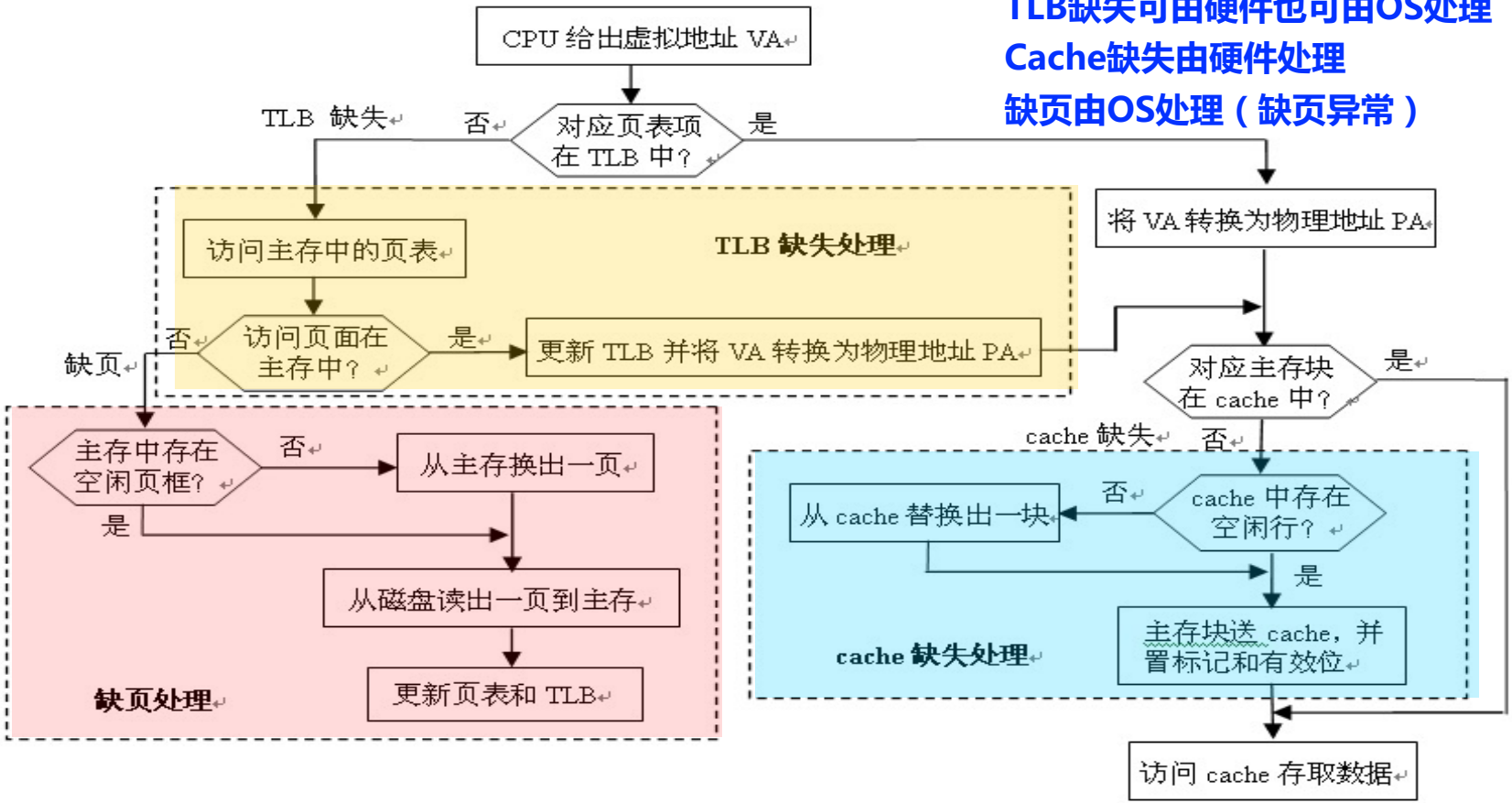
TLB和Cache的访问过程





CPU访存过程

TLB缺失可由硬件也可由OS处理
Cache缺失由硬件处理
缺页由OS处理（缺页异常）



三种不同缺失的组合：举例

表 7.3　TLB、page、cache 3 种缺失组合

序号	TLB	page	cache	说　明
1	hit	hit	hit	可能，TLB 命中则页一定命中，信息在主存，就可能在 cache 中
2	hit	hit	miss	可能，TLB 命中则页一定命中，信息在主存，但可能不在 cache 中
3	miss	hit	hit	可能，TLB 缺失但页可能命中，信息在主存，就可能在 cache 中
4	miss	hit	miss	可能，TLB 缺失但页可能命中，信息在主存，但可能不在 cache 中
5	miss	miss	miss	可能，TLB 缺失，则页也可能缺失，信息不在主存，一定也不在 cache 中
6	hit	miss	miss	不可能，页缺失，说明信息不在主存，TLB 中一定没有该页表项
7	hit	miss	hit	不可能，页缺失，说明信息不在主存，TLB 中一定没有该页表项
8	miss	miss	hit	不可能，页缺失，说明信息不在主存，cache 中一定也没有该信息

- 最好的情况是**hit、hit、hit**，此时，访问主存几次？　**不需要访问主存！**
- 以上组合中，最好的情况是？　**hit、hit、miss和miss、hit、hit**　**访存1次**
- 以上组合中，最坏的情况是？　**miss、miss、miss**　**需访问磁盘、并访存至少2次**
- 介于最坏和最好之间的是？　**miss、hit、miss**　**不需访问磁盘、但访存至少2次**

分段式虚拟存储器

- **分段系统的实现**
  - 程序员或OS将程序模块或数据模块分配给不同的主存段，一个大程序有多个代码段和多个数据段构成，是按照程序的逻辑结构划分而成的多个相对独立的部分。

    （例如，代码段、只读数据段、可读写数据段等）
  - **段通常带有段名或基地址**，便于编写程序、编译器优化和操作系统调度管理
  - **分段系统将主存空间按实际程序中的段来划分**，每个段在主存中的位置记录在**段表**中，并附以“段长”项
  - **段表由段表项组成**，段表本身也是主存中的一个可再定位段

段式虚拟存储器的地址映像

Faults (异常情况)：

缺段（段不存在）：装入位= 0

地址出界： 偏移量超出最大段长

保护违例：访问操作与访问方式不匹配

分段式虚存的地址转换

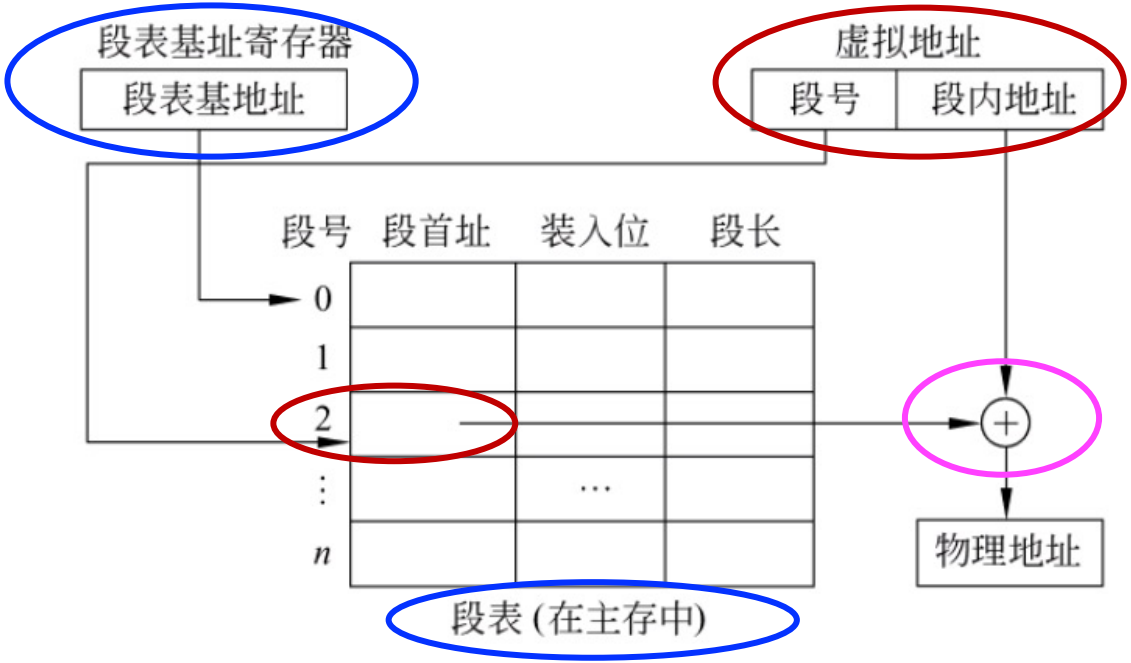


图 7.45 分段式虚存的地址转换

段页式存储器

- **段页式系统基本思想**
  - **段、页式结合**：
    - 程序的虚拟地址空间**按模块分段、段内再分页**，进入主存仍以页为基本单位
  - 逻辑地址由段地址、页地址和偏移量三个字段构成
  - 用段表和页表（每段一个）进行两级定位管理
  - 根据段地址到段表中查阅与该段相应的页表首地址，转向页表，然后根据页地址从页表中查到该页在主存中的页框地址，由此再访问到页内某数据

存储保护的基本概念

- **什么是存储保护？**
  - 为避免多道程序相互干扰，防止某程序出错而破坏其他程序的正确性或不合法地访问其他程序或数据区，应对每个程序进行存储保护
- **操作系统程序和用户程序都需要保护**
- **以下情况发生存储保护错**
  - 地址越界（转换得到的物理地址不属于可访问范围）
  - 访问越权（访问操作与所拥有的访问权限不符）
    - **页表中设定访问（存取）权限**
- **访问属性的设定**
  - 数据段可指定R/W或RO；程序段可指定R/E或RO
- **最基本的保护措施：**规定各道程序只能访问属于自己所在的存储区和共享区
  - 对于属自己存储区的信息：可读可写，只读/只可执行
  - 对共享区或已获授权的其他用户信息：可读不可写
  - 对未获授权的信息（如OS内核、页表等）：不可访问

存储保护的硬件支持

- 为了对操作系统的存储保护提供支持，硬件必须具有以下三种基本功能：
  - **支持至少两种运行模式：**
    - **管理模式(Supervisor Mode)**

      执行系统程序（内核）时处理器所处的模式称为**管理模式(Supervisor Mode)**，或称管理程序状态，简称管态、管理态、核心态、内核态
    - **用户模式(User Mode)**

      CPU执行非操作系统的用户程序时，处理器所处的模式就是**用户模式**，或称用户状态、目标程序状态，简称为目态或用户态
  - **使一部分CPU状态只能由内核程序读写而不能由用户程序读写**：这部分状态包括：User/Supervisor模式位、页表首地址、TLB等。OS内核可以用特殊的指令（一般称为**管态指令或特权指令**）来读写这些状态
  - **提供让CPU在管理模式（内核态）和用户模式（用户态）相互转换的机制**：“异常”和“陷阱”（系统调用）使CPU从用户态转到内核态；异常处理中的“返回”指令使CPU从内核态态转到用户态
- 通过上述三个功能并把页表保存在OS的地址空间，OS就可以更新页表，并防止用户程序改变页表，确保用户程序只能访问由OS分配给的存储空间

RISC-V的三种特权模式

- **机器模式（Machine Mode）**

  也称为**M模式**。M模式是RISC-V中硬件线程（hart）可以执行的最高权限模式。在M模式下运行的hart可访问所有资源和所有内存，具有启动和配置系统等系统底层支撑功能。因此，M模式是所有标准RISC-V处理器都**必须实现的运行模式**。通常，用于嵌入式系统的简单RISC-V微控制器仅支持M模式。

- **用户模式（User Mode）**

  也称为**U模式**。为阻止不可信代码通过特权指令访问系统中重要的信息或控制系统的行为，RISC-V把处理器执行普通应用程序时的模式定义为用户模式。在U模式下**只能执行各类普通指令**，不能执行特权指令。安全的嵌入式系统一般支持M模式+U模式，执行ecall指令从U→M模式，执行mret指令从M→U模式。

- **监管模式（Supervisor Mode）**

  也称为**S模式**。复杂的RISC-V系统会使用请求分页虚拟存储管理方式，对应的运行模式为支持存储保护的S模式。因此，**操作系统内核程序运行在S模式下**。

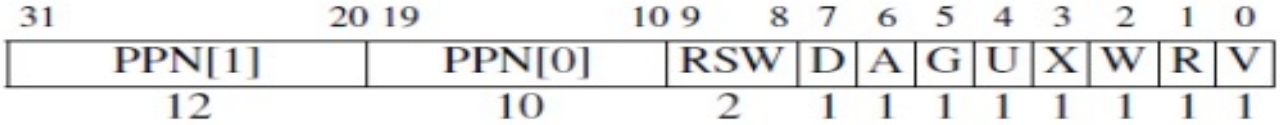
➢ 在支持M、S和U三种模式的系统中，可实现类似UNIX的操作系统。RISC-V当前的特权模式属于处理器的内部状态，软件无法直接查询。模式位占两位，00：U模式；01：S模式；11：M模式。

RISC-V不同特权模式组合系统的存储保护机制

- **RISC-V采用可灵活组合特权模式的方式来实现不同的系统。**
- **仅支持M模式的系统**
  - 因为所有程序都在M模式下执行，所以所有代码都可直接访问系统中的重要信息，因此，这种系统易受到不可信代码的控制和攻击，**只能用于安全性和可靠性要求不高的场合**，例如，只运行少数甚至一个程序的微控制器。
- **支持M模式和U模式的系统**
  - 安全的嵌入式系统一般都属于这种系统。通过将不可信代码限制在较低特权的U模式执行，来保证系统的重要信息得到保护。
  - 处理器具有**物理内存保护（Physical Memory Protection，PMP）功能**，允许M模式指定U模式可以访问的物理内存地址空间范围。
  - 在U模式下，如果处理器在取指令或执行Load/Store指令时，发生了与配置寄存器设置的权限不相符的操作，则会引起异常。
  - 只有在M模式下才能执行级别较高的特权指令，以避免不可信代码对系统的破坏。
- **支持M模式、S模式和U模式的系统**
  - S模式比U权限高，但比M低。S模式下不能使用M模式下的CSR寄存器和指令。
  - S模式支持页式虚拟存储管理机制，因而这种系统可实现现代操作系统的功能。
  - 在M模式下发生的异常总是在M模式下处理，S模式下发生的异常，只可能在M或S模式下处理，不可能移交到U模式下处理。

S模式下的页式虚拟存储管理机制

- **RV32的分页方案Sv32：4GB虚拟空间、4KB页面、页目录项（PDE）和页表项（PTE）都占32位，内容如下：**



- **V：**有效位。0表示对应页不在主存，发生缺页故障（page fault）。
- **R、W、X：** 存取权限位。表示是否可读、可写和可执行。当指令执行时实际发生的操作与存取权限发生矛盾，则会引起访存故障或取指令故障。若全为0，则说明是页目录项，其中PPN存放的是所指向的下一级页表所在的物理页（即页框）号。
- **U：**是否U模式可访问。0表示U模式不能访问该页，但S模式可以；1表示U模式下能访问该页，而默认S模式不能。若不能访问时进行了访存操作，则引起访存故障。默认S模式不能访问U=1的页面，可防止恶意程序哄骗操作系统窃取其它程序的数据。
- **G：**指定全局映射。全局映射指存在于整个地址空间的映射，通常用于OS自身使用的页面。页目录项时，G=1说明其下一级所有页表项都是全局映射。将全局映射标记为非全局映射只会降低性能，而将非全局映射标记为全局映射则会发生访存故障。
- **A：**访问位。记录自上一次A位被清除以来，该页是否被访问过。
- **D：**脏位。记录自上一次D位被清除以来，该页是否被写过。
- **RSW：**留给操作系统使用，硬件可忽略。
- **PPN：**物理页（页框）号。

Sv32中地址转换过程

- **satp（Supervisor Address Translation and Protection）寄存器是S模式下的一个CSR寄存器，其中，MODE字段可设置是否开启分页模式，以及支持多少位虚拟地址，在开启分页模式的情况下，PPN字段用于指出页目录表的页框号。**

Step1. 读取位于主存地址

(satp.PPN×4096+VA[31:22]×4)处的页目录项（PDE）

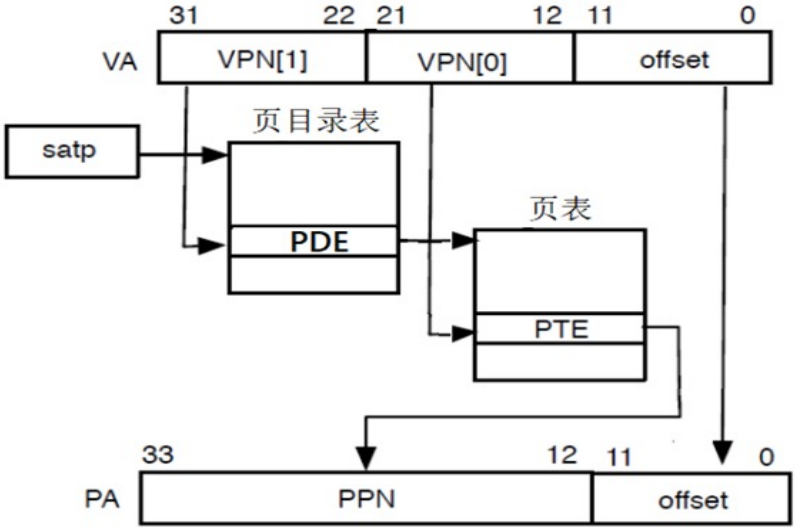
Step2. 读取位于主存地址

(PDE.PPN×4096+VA[21:12]×4)的页表项（PTE）

Step3. 页表项中22位的PPN字段和VA中的页内偏移

地址（offset）组成34位物理地址：

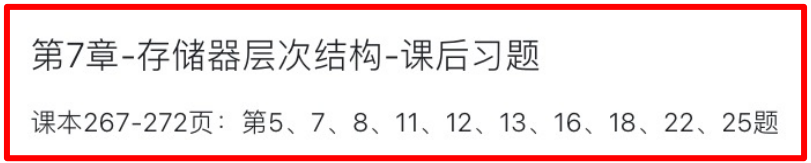
(PTE.PPN×4096+VA[11: 0])



- **地址转换过程中，处理器会根据页表项内容进行各种异常情况检查。**
  - 例如，若V=0，则引起缺页故障；若实际操作与R、W、X指定的操作不符，则引起访存故障或取指令故障（有时统称为保护错）。

课程习题（作业）——截止日期：12月16日晚23:59

- **课本267-272页**：第5、7、8、11、12、13、16、18、22、25题
- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）



- 命名：学号+姓名+第*章。
- 若提交遇到问题请及时发邮件或在下一次上课时反馈。

课程习题（作业）——截止日期：12月16日晚23:59

5. 假定用 8K×8 位的 EPROM 芯片组成 32K×16 位的只读存储器，要求回答以下问题。

（1）数据寄存器最少应有多少位？

（2）地址寄存器最少应有多少位？

（3）共需多少个 EPROM 芯片？

7. 假定一个存储器系统支持四体交叉存取，某程序执行过程中访问地址序列为 3，9，17，2，51，37，13，4，8，41，67，10，则哪些地址访问会发生体冲突？

8. 假定一个程序重复完成将磁盘上一个 4KB 的数据块读出，进行相应处理后，写回到磁盘的另外一个数据区。各数据块内信息在磁盘上连续存放，并随机地置于磁盘的一个磁道上。磁盘转速为 7200RPM，平均寻道时间为 10ms，磁盘最大数据传输率为 40MB/s，磁盘控制器的开销为 2ms，没有其他程序使用磁盘和处理器，并且磁盘读写操作和磁盘数据的处理时间不重叠。若程序对磁盘数据的处理需要 20000 个时钟周期，处理器时钟频率为 500MHz，则该程序完成一次数据块“读出-处理-写回”操作所需的时间为多少？每秒钟可以完成多少次这样的数据块操作？

课程习题（作业）——截止日期：12月16日晚23:59

11. 假定某计算机主存地址空间大小为 1GB，按字节编址，cache 的数据区（即不包括标记、有效位等存储区）有 64KB，块大小为 128 字节，采用直接映射和直写（write through）方式。请问：

（1）主存地址如何划分？要求说明每个字段的含义、位数和在主存地址中的位置。

（2）cache 的总容量为多少位？

12. 假定某计算机的 cache 共 16 行，开始为空，块大小为 1 个字，采用直接映射方式，按字编址。CPU 执行某程序时，依次访问以下地址序列：2，3，11，16，21，13，64，48，19，11，3，22，4，27，6 和 11。

（1）说明每次访问是命中还是缺失，试计算访问上述地址序列的命中率。

（2）若 cache 数据区容量不变，而块大小改为 4 个字，则上述地址序列的命中情况又如何？

13. 假定数组元素在主存按从左到右的下标优先顺序存放。试改变下列函数中循环的顺序，使得其数组元素的访问与排列顺序一致，并说明为什么修改后的程序比原来的程序执行时间更短。

```
int sum_array(int a[N][N][N])
{
    int i, j, k, sum=0;
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++) sum+=a[k][i][j];
    return sum;
}
```

课程习题（作业）——截止日期：12月16日晚23:59

16. 以下是对矩阵进行转置的程序段：

```
typedef int array[4][4];
void transpose(array dst, array src)
{
    int i, j;
    for(i=0; i<4; i++)
        for(j=0; j<4; j++) dst[j][i]=src[i][j];
}
```

假设该段程序运行的计算机中 sizeof(int)＝4，且只有一级 cache，其中 L1 数据 cache 的数据区大小为 32B，采用直接映射、回写方式，块大小为 16B，初始为空。数组 dst 从地址 0000 C000H 开始存放，数组 src 从地址 0000 C040H 开始存放。填写下表，说明对数组元素 src[row][col]和 dst[row][col]的访问是命中(hit)还是缺失(miss)。若将 L1 数据 cache 的数据区容量改为 128B，请重新填写表中内容。

	src 数组				dst 数组			
	col＝0	col＝1	col＝2	col＝3	col＝0	col＝1	col＝2	col＝3
row＝0	miss				miss			
row＝1								
row＝2								
row＝3								

课程习题（作业）——截止日期：12月16日晚23:59

18. 假设某计算机的主存地址空间大小为 64MB，采用字节编址方式。其 cache 数据区容量为 4KB，采用 4 路组相联映射方式、LRU 替换算法和回写(write back)策略，块大小为 64B。请问：

(1) 主存地址字段如何划分？要求说明每个字段的含义、位数和在主存地址中的位置。

(2) 该 cache 的总容量有多少位？

(3) 假设 cache 初始为空，CPU 依次从 0 号地址单元顺序访问到 4344 号单元，重复按此序列共访问 16 次。若 cache 命中时间为 1 个时钟周期，缺失损失为 10 个时钟周期，则 CPU 访存的平均时间为多少时钟周期？

22. 假定有 3 个处理器，分别带有以下不同的 cache：

cache 1：采用直接映射方式，块大小为 1 个字，指令和数据的缺失率分别为 4%和 6%。

cache 2：采用直接映射方式，块大小为 4 个字，指令和数据的缺失率分别为 2%和 4%。

cache 3：采用 2 路组相联映射方式，块大小为 4 个字，指令和数据的缺失率分别为 2%和 3%。

在这些处理器上运行同一个程序，其中有一半是访存指令，在 3 个处理器上测得该程序的 CPI 都为 2.0。已知处理器 1 和 2 的时钟周期都为 420ps，处理器 3 的时钟周期为 450ps。若缺失损失为(块大小＋6)个时钟周期，请问：哪个处理器因 cache 缺失而引起的额外开销最大？哪个处理器执行速度最快？

课程习题（作业）——截止日期：12月16日晚23:59

25. 假定一个计算机系统中有一个 TLB 和一个 L1 数据 cache。该系统按字节编址，虚拟地址 16 位，物理地址 12 位，页大小为 128B；TLB 采用 4 路组相联方式，共有 16 个页表项；L1 数据 cache 采用直接映射方式，块大小为 4B，共 16 行。在系统运行到某一时刻时，TLB、页表和 L1 数据 cache 中的部分内容如下：

组号	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位
0	03	—	0	09	0D	1	00	—	0	07	02	1
1	13	2D	1	02	—	0	04	—	0	0A	—	0
2	02	—	0	08	—	0	06	—	0	03	—	0
3	07	—	0	63	0D	1	0A	34	1	72	—	0

（a）TLB(4 路组相联)：4 组、16 个页表项

课程习题（作业）——截止日期：12月16日晚23:59

虚页号	页框号	有效位
00	08	1
01	03	1
02	14	1
03	02	1
04	—	0
05	16	1
06	—	0
07	07	1
08	13	1
09	17	1
0A	09	1
0B	—	0
0C	19	1
0D	—	0
0E	11	1
0F	0D	1

(b) 部分页表(开始 16 项)

行索引	标记	有效位	字节 3	字节 2	字节 1	字节 0
0	19	1	12	56	C9	AC
1	—	0	—	—	—	—
2	1B	1	03	45	12	CD
3	—	0	—	—	—	—
4	32	1	23	34	C2	2A
5	0D	1	46	67	23	3D
6	—	0	—	—	—	—
7	16	1	12	54	65	DC
8	24	1	23	62	12	3A
9	—	0	—	—	—	—
A	2D	1	43	62	23	C3
B	—	0	—	—	—	—
C	12	1	76	83	21	35
D	16	1	A3	F4	23	11
E	33	1	2D	4A	45	55
F	—	0	—	—	—	—

(c) L1 数据 cache：直接映射，共 16 行，块大小为 4B

请问(假定图中数据都为十六进制形式)：

(1) 虚拟地址中哪几位表示虚拟页号？哪几位表示页内偏移量？虚拟页号中哪几位表示 TLB 标记？哪几位表示 TLB 索引？

(2) 物理地址中哪几位表示物理页号？哪几位表示页内偏移量？

(3) 主存物理地址如何划分成标记字段、行索引字段和块内地址字段？

(4) CPU 从地址 067AH 中取出的值为多少？说明 CPU 读取地址 067AH 中内容的过程。

课程实验——截止日期：12月16日晚23:59

- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）

- 命名：学号+姓名+实验*。
- **提交：文件打包，提交ZIP压缩文件。**

课程实验——截止日期：12月16日晚23:59

实验三 Cache和程序访问的局部性

实验目的: 通过实际程序的执行结果，了解程序访问的局部性对带有Cache 的计算机系统性能的影响。
实验要求: 在以下程序中，修改或添加必要的语句(如计时函数等)，以计算和打印主体程序段的执行时间。分别以M=10000，N=100；M=1000，N=1000；M=100，N=10000，执行程序A和程序B，以比较两个程序执行时间的长短。

程序段 A assign-array-rows ( )

```
{
int i, j, a[M][N]; ……
for (i= 0; i<M; i++)
for (j= 0; j<N; j++)
a[i][j]=i+j; ……
}
```

程序段 B assign-array-cols ( )

```
{
int i, j, a[M][N]; ……
for (j= 0; j<N; j++)
for (i=0; i<M; i++) a[i][j]=i+j;
……
}
```

实验报告:

1. 给出源程序(文本文件)和执行结果。
2. 对实验结果进行分析，说明局部数据块大小、数组访问顺序等和执行时间之间的关系。

提问

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

南京大学 NANJING UNIVERSITY